Université Paris-Saclay
UVSQ - Université de Versailles Saint-Quentin-en-Yvelines

Unité d'Enseignement : Apprentissage et Réseaux de Neurones

TP 1 : Algorithme de Rétro-propagation de l'Erreur

Mise en œuvre de la Régression Logistique et du Perceptron Multi-Couches sur la base de données MNIST

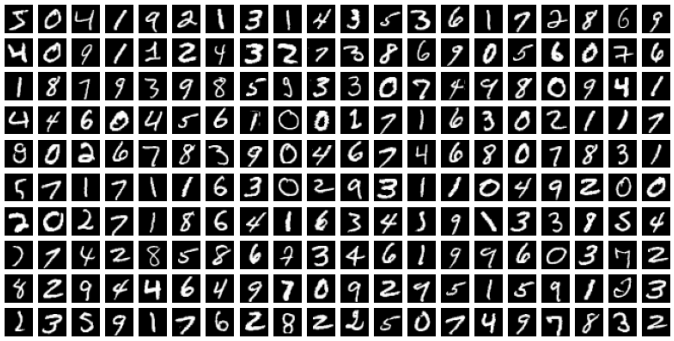
Étudiant :
DEBACHE Mounsef

Enseignants :
M. olivier pons
M. Arnaud Breloy

Février 2026

Analyse de l'espace des données

L'analyse des données de la base MNIST révèle que chaque échantillon d'apprentissage, initialement structuré sous la forme d'une matrice de 28×28 pixels, est transformé en un vecteur de caractéristiques après une opération d'aplatissement (*flattening*). Les résultats obtenus par le script confirment que la taille de ce vecteur est de 784, ce qui définit l'espace d'entrée comme un espace vectoriel de dimension $d = 784$. De plus, l'examen des valeurs extrêmes après normalisation indique un pixel minimum à 0.0 et un pixel maximum à 1.0, signifiant que les données ont été projetées depuis un domaine discret vers un domaine continu. Par conséquent, les images se situent mathématiquement au sein d'un hypercube unité de dimension 784 appartenant à l'espace des réels, noté $\mathcal{X} = [0, 1]^{784} \subset \mathbb{R}^{784}$. Cette représentation vectorielle permet de traiter les 60 000 exemples d'apprentissage comme un nuage de points dans un espace à haute dimension, facilitant ainsi le calcul des gradients nécessaire à l'optimisation du modèle.



Analyse de la structure et de la convexité du modèle

Architecture et état initial

L'examen structurel du modèle de régression logistique révèle une architecture composée d'une matrice de poids W de taille 784×10 et d'un vecteur de biais b de dimension 10. L'analyse numérique confirme que le système doit optimiser un total de 7 850 paramètres ajustables pour lier l'espace d'entrée des pixels à l'espace de sortie des classes. À l'état initial, avec des poids fixés à zéro, le modèle affiche une valeur de coût de $2,3026$, ce qui correspond mathématiquement à $-\ln(1/10)$. Ce résultat est parfaitement cohérent avec une distribution de probabilité uniforme où chaque classe possède une probabilité de $0,10$, démontrant que le modèle démarre dans un état d'incertitude maximale avant toute mise à jour des paramètres.

Calcul et justification des paramètres

Le nombre total de paramètres du modèle de régression logistique est déterminé par la structure de la transformation linéaire $\hat{s}_i = x_i W + b$.

— **Poids synaptiques (W) :** Chaque image d'entrée étant aplatie en un vecteur de dimension $d = 784$, et le réseau devant prédire des scores pour $K = 10$ classes, la matrice de projection W appartient à $\mathbb{R}^{784\times10}$. Elle totalise donc $784 \times 10 = 7\,840$ coefficients.

— **Biais (b) :** Pour permettre au modèle de translater les frontières de décision indépendamment de l'origine, un vecteur de biais $b \in \mathbb{R}^{1\times10}$ est ajouté, soit 10 paramètres supplémentaires.

Le nombre total de paramètres ajustables est donc de $7\,840+10 = \mathbf{7\,850}$. Ce volume de paramètres reste relativement faible par rapport à la taille de la base d'apprentissage ($N = 60\,000$ images), ce qui limite les risques de surapprentissage pour ce modèle linéaire.

Optimisation du modèle : Démonstration des gradients

Étape 1 : Gradient par rapport aux scores $\hat{s}_i$

L'objectif est de montrer que pour un exemple i et une classe donnée c, le gradient de la perte par rapport au score d'entrée du softmax est :

$$\frac{\partial L_i}{\partial s_{i,c}} = \hat{y}_{i,c} - y^*_{i,c}$$

Démonstration : En appliquant la règle de dérivation en chaîne (*chain rule*) sur la fonction de perte d'entropie croisée $L_i = -\sum_k y^*_{i,k} \ln(\hat{y}_{i,k})$ et la fonction Softmax $\hat{y}_{i,c} = \frac{e^{s_{i,c}}}{\sum_k e^{s_{i,k}}}$, nous obtenons :

$$\frac{\partial L_i}{\partial s_{i,c}} = \sum_k \frac{\partial L_i}{\partial \hat{y}_{i,k}} \frac{\partial \hat{y}_{i,k}}{\partial s_{i,c}}$$

En utilisant les propriétés de la dérivée du Softmax, à savoir $\frac{\partial \hat{y}_{i,k}}{\partial s_{i,c}} = \hat{y}_{i,c}(1 - \hat{y}_{i,c})$ lorsque $k = c$ et $\frac{\partial \hat{y}_{i,k}}{\partial s_{i,c}} = -\hat{y}_{i,k}\hat{y}_{i,c}$ lorsque $k \neq c$, le développement devient :

$$\frac{\partial L_i}{\partial s_{i,c}} = -\frac{y^*_{i,c}}{\hat{y}_{i,c}} [\hat{y}_{i,c}(1 - \hat{y}_{i,c})] - \sum_{k \neq c} \frac{y^*_{i,k}}{\hat{y}_{i,k}} [-\hat{y}_{i,k}\hat{y}_{i,c}]$$

En simplifiant les termes :

$$\frac{\partial L_i}{\partial s_{i,c}} = -y^*_{i,c} + y^*_{i,c}\hat{y}_{i,c} + \sum_{k \neq c} y^*_{i,k}\hat{y}_{i,c}$$

$$\frac{\partial L_i}{\partial s_{i,c}} = \hat{y}_{i,c} \left(y^*_{i,c} + \sum_{k \neq c} y^*_{i,k} \right) - y^*_{i,c}$$

Comme la somme des labels *one-hot* $\sum_k y^*_{i,k} = 1$, nous obtenons :

$$\frac{\partial L_i}{\partial s_{i,c}} = \hat{y}_{i,c} - y^*_{i,c}$$

D'où la forme vectorielle synthétique : $\delta y_i = \hat{y}_i - y^*_i$.

Étape 2 : Gradients par rapport aux paramètres W et b

En étendant ce résultat local à l'ensemble du batch de données de taille N, nous pouvons déduire les gradients globaux nécessaires à la mise à jour des paramètres :

1. **Gradient par rapport à W :** Par application de la règle de la chaîne matricielle $\frac{\partial L}{\partial W} = X^T \frac{\partial L}{\partial S}$, nous obtenons :
$$\frac{\partial L}{\partial W} = \frac{1}{N} X^T (\hat{Y} - Y^*) = \frac{1}{N} X^T \Delta y \tag{1}$$
2. **Gradient par rapport à b :** Puisque le biais est distribué sur l'ensemble des exemples, son gradient correspond à la moyenne des erreurs du batch sur chaque classe :
$$\frac{\partial L}{\partial b} = \frac{1}{N} \sum_{i=1}^{N} (\hat{y}_i - y^*_i) \tag{2}$$

Analyse de la convexité suite à la démonstration

Lien entre gradient et convexité

L'analyse de la dérivation obtenue à l'étape précédente permet de renforcer les conclusions sur la convexité du modèle. La forme du gradient $\delta y_i = \hat{y}_i - y_i^*$ est caractéristique des modèles linéaires utilisant une fonction de perte de type entropie croisée associée à une activation Softmax. Cette simplicité mathématique n'est pas fortuite : elle découle du fait que la fonction de coût est globalement convexe par rapport aux paramètres W et b.

Propriétés fondamentales déduites :

- **Absence de minima locaux :** Puisque la fonction est convexe, tout point stationnaire où le gradient s'annule ($\nabla L = 0$) est obligatoirement un minimum global. Cela signifie qu'une fois que la différence entre la prédiction $\hat{y}$ et la cible y^* est minimisée, nous avons atteint la meilleure solution possible pour ce modèle.
- **Stabilité de l'optimisation :** La convexité garantit que le vecteur gradient pointe toujours vers une zone de perte inférieure. Contrairement aux modèles non-convexes (comme le MLP), nous ne risquons pas de rester bloqués dans des "cuvettes" de l'espace des paramètres qui ne représenteraient pas l'optimum.
- **Impact du pas d'apprentissage :** Dans ce paysage d'erreur en forme de "bol", le choix de η influe uniquement sur la vitesse à laquelle on atteint le fond, et non sur la capacité du modèle à trouver le minimum global.

Conclusion sur la garantie de convergence

En conclusion, la démonstration mathématique des gradients confirme que la régression logistique bénéficie d'une structure d'optimisation idéale. Tant que le pas d'apprentissage est suffisamment petit pour ne pas diverger, l'algorithme de descente de gradient convergera mathématiquement vers l'unique solution optimale, assurant ainsi la robustesse du classifieur sur la base MNIST.

Analyse de l'influence du pas d'apprentissage (η)

Commentaire sur la convergence :

L'analyse visuelle des poids confirme l'impact du pas d'apprentissage sur la dynamique d'optimisation. On observe qu'un pas d'apprentissage plus petit ($\eta = 0.01$) produit des motifs très lisses et peu contrastés, car les mises à jour des paramètres à chaque itération sont de faible amplitude. Cela démontre expérimentalement qu'un pas réduit nécessite un nombre d'époques nettement plus important pour converger vers un état stable où les caractéristiques discriminantes des chiffres sont clairement identifiées.

Il est important de noter que l'aspect "flou" des poids observés pour $\eta = 0.01$ est une conséquence directe d'une convergence inaboutie après seulement 20 époques. En augmentant significativement le nombre d'itérations, les structures des poids gagneraient en netteté et en détails, finissant par rejoindre l'optimum atteint plus rapidement par le pas de 0.1.

Cependant, dans un contexte de performance computationnelle, le choix d'un pas plus élevé ($\eta = 0.1$) est privilégié car il permet d'extraire les caractéristiques discriminantes des chiffres (les "templates" rouges et bleus) en un temps réduit. Un pas trop petit, bien que plus stable théoriquement, impose un coût de calcul inutile pour un problème dont la surface de coût est convexe et bien conditionnée comme celle de la régression logistique sur MNIST.

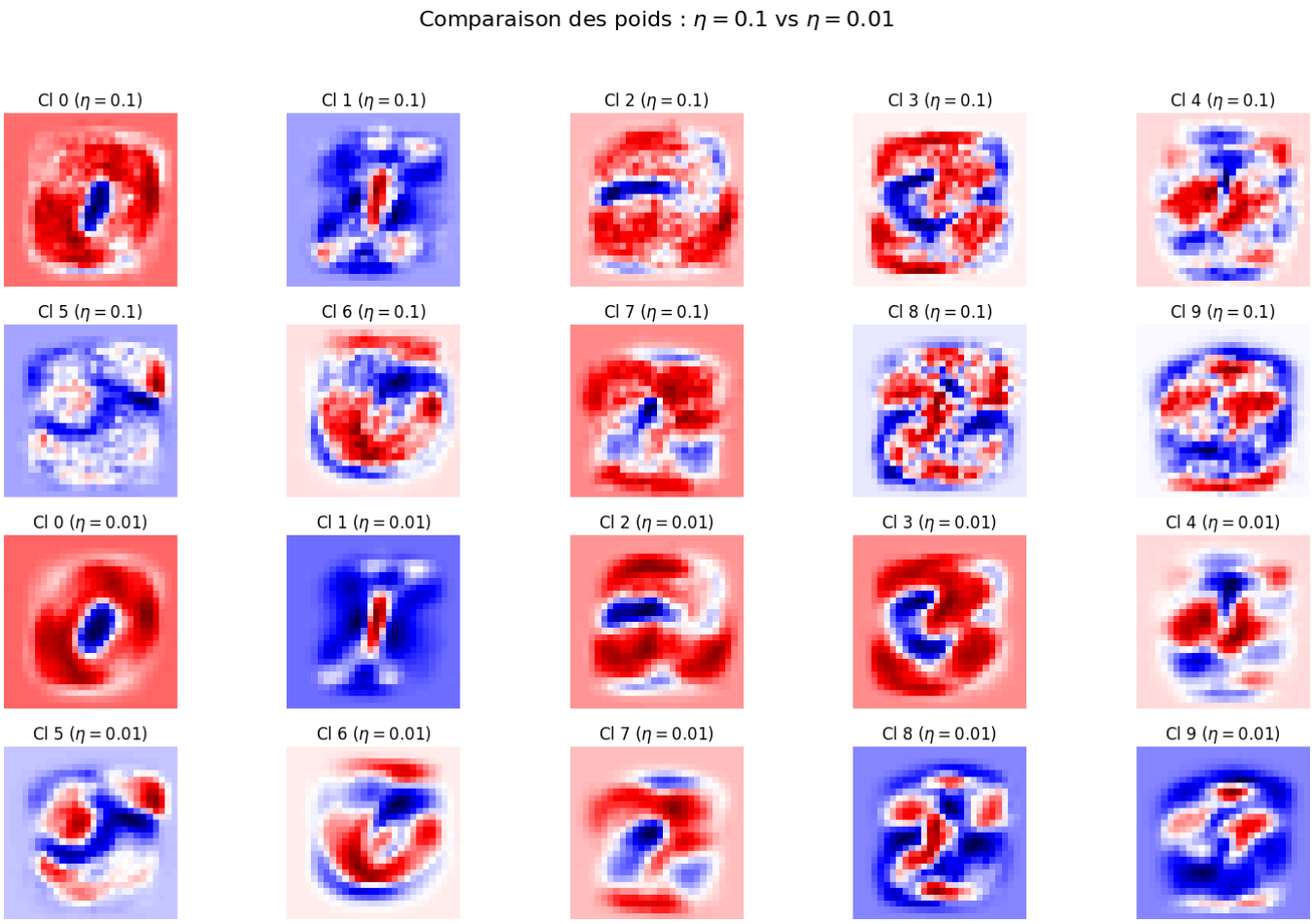


FIGURE 1 – Comparaison des matrices de poids W après 20 époques : $\eta = 0.1$ (lignes du haut) vs $\eta = 0.01$ (lignes du bas).

Perceptron Multi-Couches (MLP)

Question : La fonction de coût de l'Eq. (3) est-elle convexe par rapport aux paramètres du modèle ? Peut-on assurer la convergence vers le minimum global ?

Analyse de la convexité

Bien que la fonction de coût utilisée reste l'entropie croisée définie à l'équation (3), sa nature mathématique change radicalement avec l'architecture du Perceptron. Dans ce modèle, les paramètres à optimiser incluent désormais les poids et biais de la couche cachée (W_h, b_h) ainsi que ceux de la couche de sortie (W_y, b_y). L'introduction de la non-linéarité sigmoïde entre ces couches brise la propriété de convexité que nous avions dans le modèle linéaire.

En conséquence, la fonction de coût n'est plus convexe par rapport aux paramètres du MLP. Le paysage de l'erreur devient irrégulier, ce qui entraîne des conséquences majeures pour l'optimisation : même avec un pas d'apprentissage parfaitement choisi, il est impossible d'assurer la convergence vers le minimum global. L'algorithme de descente de gradient peut se retrouver piégé dans un minimum local, stagner sur un point selle ou ralentir considérablement dans des zones de faible pente (zones plates). La performance finale du réseau devient alors dépendante de l'initialisation des poids.

Évaluation des stratégies d'initialisation du MLP

Présentation synthétique des résultats

Afin d'évaluer l'impact de la condition initiale sur l'apprentissage d'un réseau à couche cachée, nous avons testé trois méthodes d'initialisation des poids. Le tableau ci-dessous récapitule les performances finales obtenues après 100 époques d'entraînement avec un pas de gradient $\eta = 1.0$.

Type d'initialisation	**Accuracy Finale**	**Observation sur la convergence**
Initialisation à zéro	63,96 %	Stagnation précoce
Loi Normale ($\sigma = 0.1$)	97,84 %	Apprentissage performant
Initialisation de Xavier	**98,07 %**	**Convergence optimale**

TABLE 1 – Comparaison des performances du MLP selon l'initialisation des paramètres.

Analyse des résultats et conclusions

1. Initialisation à zéro

L'initialisation des matrices à zéro produit une précision médiocre de **63,96 %**. Contrairement à la régression logistique où cette initialisation permettait d'atteindre 92 %, elle s'avère ici inefficace pour un réseau multicouche.

Conclusion : Ce résultat s'explique par le fait que tous les neurones de la couche cachée reçoivent exactement le même signal et le même gradient lors de la rétro-propagation. Bien que l'on observe une progression de l'accuracy de 10 % à environ 64 %, prouvant qu'une minimisation de la fonction de coût a bien lieu, le modèle reste structurellement limité. Puisque les neurones évoluent de façon identique, le réseau est incapable de "briser la symétrie" pour apprendre des caractéristiques variées. En pratique, l'absence d'initialisation aléatoire empêche la spécialisation des unités cachées, forçant le MLP à se comporter comme un modèle linéaire simple. Le réseau ne peut donc pas exploiter la richesse de sa couche cachée pour capturer des frontières de décision complexes, ce qui explique le plafonnement des performances bien en dessous des 98 % attendus.

2. Initialisation Normale

En utilisant une loi normale avec un écart-type $\sigma = 0.1$, les performances augmentent radicalement pour atteindre **97,84 %**.

Conclusion : L'introduction de valeurs aléatoires permet de briser la symétrie initiale entre les neurones. Chaque unité de la couche cachée commence à se spécialiser dans la détection de motifs différents dès les premières itérations. On observe une convergence rapide, dépassant largement les capacités du modèle linéaire simple.

3. Initialisation de Xavier

L'initialisation de Xavier fournit le meilleur résultat avec **98,07 %** de précision.

Conclusion : Cette méthode, qui ajuste la variance des poids en fonction du nombre de neurones en entrée ($1/\sqrt{n_i}$), permet de maintenir une dynamique de signal stable à travers les couches. Elle évite que les signaux ne s'atténuent ou ne s'amplifient de manière excessive (problème de disparition du gradient). C'est cette stabilité qui permet au modèle d'atteindre et de dépasser l'objectif des 98 % sur la base MNIST.

Conclusion Générale

Ce travail pratique a permis de consolider la compréhension théorique et technique des réseaux de neurones par une implémentation rigoureuse, *from scratch*, de la régression logistique et du Perceptron Multi-Couches (MLP). L'étude de la **régression logistique** a illustré l'efficacité d'un modèle linéaire dans un espace de grande dimension ($d = 784$), tout en démontrant l'intérêt de la **convexité** de sa fonction de coût qui assure une convergence systématique vers un minimum global. À l'opposé, le passage au **MLP** a révélé la puissance des frontières de décision non linéaires, permettant de franchir le cap des **98 %** de précision. Cette performance accrue s'accompagne toutefois d'une perte de convexité, rendant l'optimisation dépendante de facteurs critiques tels que l'**initialisation des poids**. L'expérience a prouvé que des méthodes avancées comme celle de **Xavier** sont indispensables pour briser la symétrie des neurones et stabiliser la propagation des signaux. Enfin, l'analyse du **pas d'apprentissage** (η) a mis en évidence le compromis nécessaire entre vitesse de convergence et stabilité numérique. En somme, ce TP démontre que la maîtrise combinée de l'architecture, de la stratégie d'initialisation et de la dynamique du gradient est la clé pour transformer un simple classifieur linéaire en un approximateur universel performant, capable de traiter efficacement la complexité des données réelles comme celles de la base MNIST.